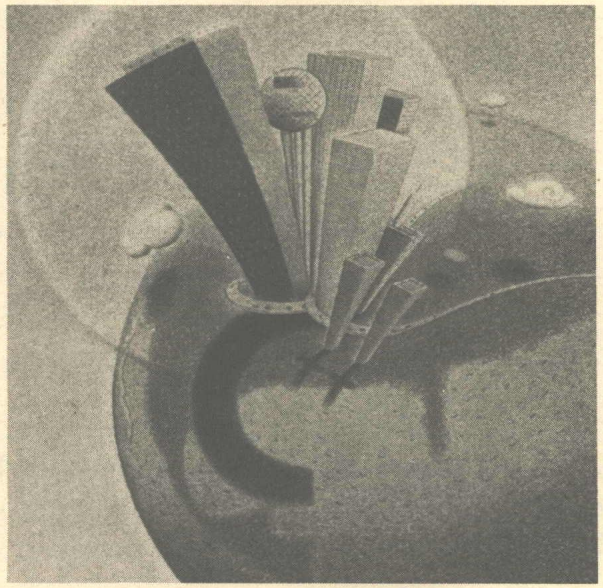
สุทธิศักดิ์ พงศ์ธนาพาณิช

เปิดโลกการเข้ารหัสข้อมูลด้วยภาษาซี ตอน Arithmetic Coding with Order-1 Modeling



เสนอวิธีการเข้ารหัสข้อมูลแบบ Arithmetic coding เพื่อเพิ่มประสิทธิภาพในการบีบอัดข้อมูล

ในตอนนี้จะนำวิธีการนำโมเดลลำดับที่ 1 มาใช้กับการเข้ารหัสข้อมูลแบบ Arithmetic coding เพื่อเพิ่มประสิทธิภาพในการบีบอัดข้อมูลให้สูงขึ้น หลังจากผู้อ่านได้ทราบถึงเทคนิคการโปรแกรมของโมเดลลำดับที่ 1 ผู้อ่านก็สามารถนำไปประยุกต์ต่อไปเพื่อสร้างโปรแกรมที่สามารถใช้โมเดลลำดับที่ 2, 3 หรือสูงขึ้นเรื่อย ๆ ได้อย่างไม่ยากนัก

พื้นฐานของโมเดลลำดับที่ 1

จากที่เคยเขียนเอาไว้ในส่วนที่เป็นทฤษฎีของเทคนิคการเข้ารหัสด้วยวิธี Huffman หรือ Arithmetic ผู้อ่านก็จะทราบว่าประสิทธิภาพในการบีบอัดข้อมูลจะดีหรือไม่นั้น ส่วนหนึ่งก็ขึ้นกับโมเดลที่เรานำมาใช้ในการคำณวนด้านสถิติของสตริงที่จะถูกเขารหัส หรืออธิบายง่าย ๆ ก็คือ ถ้าหากเราสามารถทำนายค่าความถี่หรือความน่าจะเป็นของการมีตัวอักษร ascii ในสายสตริงได้อย่างถูกต้องแม่นยำ ก็จะทำให้การบีบอัดข้อมูลได้ผลลัพธ์ที่ดีที่สุดเลยทีเดียว ทั้งนี้เนื่องจากเทคนิคการเข้ารหัสข้อมูลข้างต้นมีพื้นฐานอยู่บนการแทนที่สตริงที่มีความถี่สูงสุดด้วยจำนวนบิตที่ต่ำสุดนั่นเอง ด้วยเหตุนี้เองนักคณิตศาสตร์จึงได้มีความคิดที่จะพัฒนาอัลกอริทึมของโมเดลเพื่อให้สามารถทำนายค่าความถี่หรือความน่าจะเป็นของข้อมูลในสตริงสูงขึ้น จึงเป็นที่มาของการทำโมเดลลำดับที่ 1 หรือสูงขึ้น

การที่เราจะบอกว่ามีการนำโมเดลลำดับที่เท่าใดมาใช้นั้น เราก็จะดูจากจำนวนตัวอักษรผ่านมาที่ถูกนำมาใช้ในการคำนวณค่าความถี่ของข้อมูลตัวอักษรปัจจุบัน ซึ่งโมเดลลำดับที่ 0 ก็หมายถึง การคำนวณค่าความถี่ของข้อมูลตัวอักษรปัจจุบันโดยไม่มีการนำตัวอักษรที่ผ่านมามาใช้ และโมเดลลำดับที่ 1 ก็หมายถึง การคำนวณค่าความถี่ของข้อมูลตัวอักษรปัจจุบัน โดยมีการนำตัวอักษรที่ผ่านมา 1 ตัวอักษรมาใช้ ดังนั้นในการเขียนโค้ดสำหรับโมเดลลำดับที่ 1 จึงต้องมีการสร้างตารางความถี่สำหรับข้อมูลทุกตัวถึง 256 ตารางเลยทีเดียว (สำหรับแอสกี 0 ถึง 255) และโมเดลลำดับที่ 2 ก็ต้องมีการสร้างตารางความถี่สำหรับข้อมูลถึง 65,536 ตาราง ซึ่งสามารถเขียนเป็นสมการเพื่อแสดงความสัมพันธ์ระหว่างลำดับที่ของโมเดลและจำนวนตารางได้ดังนี้

$$TblNum = 256^{OrderN}$$

โดยที่ TblNum หมายถึง จำนวนตารางค่าความถี่

OrderN หมายถึง ลำดับที่ของโมเดล ตั้งแต่ 0, 1, 2, ...

ด้วยเหตุนี้ผู้อ่านจะเห็นว่าถ้าหากเรามีการสร้างโปรแกรมบีบอัดข้อมูล เราคงจะไม่ใช้โมเดลที่มีลำดับมากกว่า 1 ทั้งนี้เนื่องจากการสร้างตารางค่าความถี่มาก ๆ เช่น 65,536 ตารางนั้นจำเป็นต้องใช้หน่วยความจำและความ

สามารถในการประมวลผลของซีพียูเป็นอย่างมาก ซึ่งถ้าโปรแกรมทำงานช้ามาก ๆ ก็จะไม่สามารถนำโปรแกรมมาพัฒนาต่อในเชิงธุรกิจ ดังนั้นในความคิดเห็นส่วนตัวผู้เขียนคิดว่าสำหรับโมเดลที่จะนำมาใช้กับเทคนิคการบีบอัดข้อมูลด้วยวิธี Huffman หรือ Arithmetic นั้นควรจะเป็นโมเดลลำดับที่ 0 หรือ 1 เท่านั้น

เหตุผลง่าย ๆ ที่เรานำตัวอักษรที่ผ่านมาใช้ในการคำนวณค่าความถี่นั้น ก็ต้องถือว่าเริ่มมาจากความพยายามที่จะนำเทคนิคที่เรียกว่า adaptive มาใช้ ซึ่งเทคนิค adaptive นี้จะไม่มีการอ่านข้อมูลจากไฟล์ทีเดียวทั้งหมดเพื่อนำมาคำนวณความถี่ เพราะจะทำให้เสียเวลาเป็นอย่างมาก แต่จะเป็นการอ่านสายสตริงเป็นช่วง ๆ จากไฟล์ และทำการปรับปรุงโมเดลทุกครั้งที่พบตัวอักษร ดังนั้นถ้าหากเรานำตัวอักษรที่ผ่านมาใช้ในการคำนวณค่าความถี่ก็เท่ากับว่าเราได้ทำการเพิ่มจำนวนตัวอักษรในแต่ละสายสตริงขึ้นมาทันที และที่สำคัญตัวอักษรที่เพิ่มขึ้นมานี้ก็มีความเป็นไปได้ที่จะซ้ำมากที่สุดเช่นกัน ซึ่งทำให้เราสรุปได้ว่าการใช้โมเดลลำดับสูง ๆ นั้นจะทำให้อัตราส่วนการบีบอัดข้อมูลเพิ่มสูงขึ้นเช่นกัน

เทคนิค adaptive นี้เป็นเทคนิคที่เหมาะสมเป็นอย่างมากสำหรับโมเดลลำดับสูง ๆ เพราะแต่เดิมในการบีบอัดข้อมูลด้วยวิธี huffman นั้นจะประกอบด้วย 2 ขั้นตอนหลัก ก็คือการนับค่าความถี่ของแต่ละตัวอักษรในสายสตริง และการนำค่าความถี่มาใช้เข้ารหัสตัวอักษร ดังนั้นเพื่อให้รูทีนที่ทำการขยายขนาดข้อมูลหรือถอดรหัสสามารถทำงานได้อย่างถูกต้องผู้อ่านก็จำเป็นต้องจัดเก็บตารางความถี่ลงในไฟล์ เพื่อให้รูทีนที่ทำการขยายขนาดข้อมูลสามารถทำการสร้างโมเดลได้ต่อไป ดังนั้นถ้าหากขนาดของไฟล์ที่จะบีบอัดมีขนาดเล็กกว่าตารางความถี่ ซึ่งในกรณีนี้ก็จะทำให้ไฟล์ที่ถูกบีบอัดข้อมูลมีขนาดใหญ่ขึ้น (น่าจะเรียกว่าการเพิ่มขนาดข้อมูลมากกว่า) ดังนั้นเมื่อผู้อ่านนำเทคนิค adaptive มาใช้ก็จะสามารถแก้ไขปัญหาข้างต้นได้ทันที เพราะเทคนิค adaptive จะทำให้รูทีนบีบอัดและขยายขนาดข้อมูลจะเริ่มต้นทำงานด้วยโมเดลเดียวกัน และจะทำการปรับปรุงโมเดลทุกครั้งที่มีการอ่านตัวอักษรใหม่เข้ามา ดังนั้นรูทีนที่ขยายขนาดข้อมูลจึงไม่มีความจำเป็นที่จะต้องทราบตารางค่าความถี่ในตอนเริ่มต้น

อย่างไรก็ตาม เทคนิค adaptive ก็ยังคงมีข้อเสียเหมือนกัน ก็คือการเริ่มต้นบีบอัดข้อมูลจะให้ผลลัพธ์ที่ไม่ดีนัก และใช้เวลาในการปรับปรุงโมเดลพอสมควร โดยเฉพาะโมเดลลำดับสูง ๆ เพราะโปรแกรมจะต้องทำการคำนวณค่าความถี่ให้กับทุก ๆ ตารางที่เกี่ยวข้องของแต่ละตัวอักษร แต่ถ้าหากผู้อ่านเปรียบเทียบกับการสูญเสียพื้นที่ในไฟล์สำหรับการจัดเก็บตารางค่าความถี่แล้วละก็ เทคนิค adaptive ยังคงเป็นเทคนิคที่ดีกว่าอย่างแน่นอนที่สุดเสมอ

การทำงานของโปรแกรม ARITH1.C

เนื่องจากโปรแกรมตัวอย่างสำหรับบทความนี้ยังคงใช้เทคนิค arithmetic ในการเข้ารหัสข้อมูล ดังนั้นผู้เขียนจึงขอไม่อธิบายรูทีนที่ทำงานในขั้นตอนการบีบอัดข้อมูลนี้ โดยผู้อ่านสามารถดูรายละเอียดได้จากตอนที่ผ่านมา แต่ผู้อ่านจะอธิบายเฉพาะในส่วนของการสร้างโมเดลลำดับที่ 1 เท่านั้น ซึ่งจะเป็นขบวนการที่มีการทำงานเหมือนกันทั้งสำหรับการบีบอัดและขยายขนาดข้อมูล

การบีบอัดข้อมูลจะถูกจัดการภายในรูทีน CompressFile โดยมีรายละเอียดดังนี้

```
context = 0;
initialize_model();
initialize_arithmetic_encoder();
for ( ; ; ) {
    c = getc( input );
    if ( c == EOF )
        c = END_OF_STREAM;
    convert_int_to_symbol( c, context, &s );
    encode_symbol( output, &s );
    if ( c == END_OF_STREAM )
      break;
    update_model( c, context );
    context = c;
}
flush_arithmetic_encoder( output );
```

การทำงานจะเริ่มต้นด้วยการกำหนดค่าเริ่มต้นให้กับตัวแปรพอยน์เตอร์ totals สำหรับตารางค่าความถี่จำนวน 256 ตารางเสียก่อน ซึ่งจะถูกจัดการภายในรูทีน initialize_model ดังนี้

```
  for ( context = 0 ; context < END_OF_STREAM ;
context++ ) {
    totals[ context ] = (int *) calloc(
END_OF_STREAM + 2, sizeof(int) );
    if ( totals[ context ] == NULL )
       fatal_error( "Failure allocating context
%d", context );
    for ( i = 0 ; i <= ( END_OF_STREAM + 1 ) ; i++ )
       totals[ context ][ i ] = I;
```

จากนั้นก็ทำการกำหนดค่าต่ำสุดและค่าสูงสุดเท่าที่จะเป็นไปได้ในการเข้ารหัสด้วยวิธี Arithmetic ให้กับตัวแปร low และ high ตามลำดับ และรีเซตให้ค่าจำนวนของบิต underflow เท่ากับ 0 ดังนี้

```
low = 0;
high = 0xffff;
underflow_bits = 0;
```

จากนั้นก็เริ่มขั้นตอนการอ่านสายสตริงเพื่อทำการสร้างโมเดล โดยเราจะทำการอ่านข้อมูลจากไฟล์ทีละตัวอักษร (1 ไบต์) และแปลงค่าแอสกีให้เป็นค่าสำหรับสัญลักษณ์ในการเข้ารหัสด้วยรูทีน convert_int_to_symbol ซึ่งขั้นตอนในรูทีนนี้ต่างจากที่ปรากฏในโปรแกรมตอนที่ผ่านมาตรงที่ เราใช้ตัวอักษรที่ผ่านมาใช้ด้วยโดยผ่านทางพารามิเตอร์ context ดังนี้

```
s->scale = totals[ context ][ END_OF_STREAM + 1 ];
s->low_count = totals[ context ][ c ];
s->high_count = totals[ context ][ c + 1 ];
```

ขั้นตอนต่อไปก็เป็นการเข้ารหัสสัญลักษณ์ที่ได้ด้วยรูทีน encode_symbol ซึ่งข้อมูลของสัญลักษณ์ เช่น ค่าต่ำ ค่าสูงและขอบเขตลำดับชั้น จะส่งผ่านมายังรูทีนนี้ผ่านโครงสร้าง SYMBOL และการทำงานภายในรูทีนจะเริ่มต้นด้วยการ สเกลค่าต่ำ ค่าสูง

```
range = (long) ( high-low ) + 1;
high = low + (unsigned short int)
      (( range * s->high_count ) / s->scale - 1 );
low = low + (unsigned short int)
      (( range * s->low_count ) / s->scale );
```

จากนั้นก็ทำการวนลูปเพื่อทำการทดสอบค่าบิตลำดับสูงของค่าต่ำและค่าสูง โดยถ้าหากบิตเซตก็ทำการเขียนบิตลงในไฟล์ได้ทันที หรือแสดงว่าเกิดบิต underflow ก็ให้ทำการเพิ่มจำนวนบิตดังกล่าวและปรับค่าของค่าต่ำและค่าสูงเสียใหม่ ดังนี้

```
  for ( ; ; ) {
    if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
      OutputBit( stream, high & 0x8000 );
      while ( underflow_bits > 0 ) {
        OutputBit( stream, ~high & 0x8000 );
        underflow_bits--;
      }
    }
    else if ( ( low & 0x4000 ) && !( high & 0x4000 ))
{
      underflow_bits += 1;
      low &= 0x3fff;
      high |= 0x4000;
    } else
      return ;
    low <<= 1;
    high <<= 1;
    high |= 1;
  }
}
```

และทำการปรับปรุงโมเดลด้วยรูทีน update_model ซึ่งภายในรูทีนนี้เพิ่มค่าความถี่ในตาราง totals ทุกครั้งหลังจากทำการเข้ารหัส ดังนี้

```
  for ( i = symbol + 1 ; i <= ( END_OF_STREAM + 1 ) ; i++ )
    totals[ context ][ i ]++;
  if ( totals[ context ][END_OF_STREAM+1 ]<MAXIMUM_SCALE )
    return;
  for ( i = 1 ; i <= ( END_OF_STREAM + 1 ) ; i++ ) {
    totals[ context ][ i ] /= 2;
    if ( totals[ context ][ i ] <= totals[ context ][ i - 1 ] )
      totals[ context ][ i ] = totals[ context ][ i-1 ]+1;
  }
```

และเมื่อสิ้นสุดการอ่านข้อมูลจากไฟล์ก็ทำการเขียนบิตที่เหลือลงในไฟล์ด้วยรูทีน flush_arithmetic_encoder ก็เป็นอันสิ้นสุดขบวนการเข้ารหัสข้อมูลด้วยวิธี Arithmetic โดยใช้โมเดลลำดับที่ 1 ดังนี้

```
  OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits-- > 0 )
      OutputBit( stream, ~low & 0x4000 );
    OutputBits( stream, 0L, 16 );
}
```

ส่วนการถอดรหัสข้อมูลก็จะถูกจัดการภายในรูทีน ExpandFile ดังนี้

```
    context = 0;
    initialize_model();
    initialize_arithmetic_decoder( input );
for ( ; ; ) {
    get_symbol_scale( context, &s );
    count = get_current_count( &s );
    c = convert_symbol_to_int( count, context, &s );
    remove_symbol_from_stream( input, &s );
    if ( c == END_OF_STREAM )
        break;
    putc( (char) c, output );
    update_model( c, context );
    context = c;
}
```

ซึ่งเริ่มต้นการทำงานเช่นเดียวกับการบีบอัดข้อมูลด้วยการกำหนดค่าเริ่มต้นให้กับตัวแปรพอยน์เตอร์ totals สำหรับตารางค่าความถี่จำนวน 256 ตาราง ด้วยรูทีน initialize_model จากนั้นก็ทำการกำหนดค่าเริ่มต้นให้กับค่าต่ำและค่าสูงและอ่านบิตจากไฟล์จำนวน 16 บิตให้กับตัวแปร code สำหรับการสร้างโมเดลแบบ adaptive ต่อไป ซึ่งจะถูกจัดการด้วยรูทีน initialize_arithmetic_decoder ดังนี้

```
    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
```

จากนั้นก็เป็นการวนลูปเพื่ออ่านสายข้อมูลขึ้นมาเพื่อทำการขยายขนาด โดยภายในวนลูปนี้จะเริ่มต้นด้วยการอ่านค่าที่ทำการสเกลของข้อมูลปัจจุบันให้กับตัวแปร s->scale ดังนี้

```
s->scale = totals[ context][ END_OF_STREAM + 1 ];
```

และทำการคำนวณย้อนกลับเพื่อหาค่าความถี่ของสัญลักษณ์ปัจจุบัน เพื่อค้นหาสัญลักษณ์ที่จะถูกขยายขนาดเป็นตัวต่อไป ซึ่งการทำงานขั้นตอนนี้จะถูกจัดการภายในรูทีน get_current_count ดังนี้

```
range = (long) ( high - low ) + 1;
count = (short int) ((((long) ( code - low ) + 1 )
* s->scale-1 ) / range );
return( count );
```

และทำการแปลงค่ารหัสของสัญญลักษณ์ที่อยู่ในช่วงค่าต่ำและค่าสูง กลับเป็นเลขจำนวนเต็มด้วยรูทีน convert_symbol_to_int ดังนี้

```
for ( c = 0; count >= totals[ context ][ c + 1 ] ; c++ )
  ;
s->high_count = totals[ context ][ c + 1 ];
s->low_count = totals[ context ][ c ];
return( c );
```

และเมื่อสามารถถอดรหัสข้อมูลได้แล้วก็ทำการลบจำนวนบิตเหล่านั้นออกจากสายสตริง เพื่อเตรียมพร้อมสำหรับการอ่านสายสตริงถัดไปด้วยรูทีน remove_symbol_from_stream โดยเริ่มต้นด้วยการคำนวณค่าต่ำและค่าสูง

```
range     = (long)( high - low ) + 1;
high      = low + (unsigned short int)
          (( range * s->high_count ) / s->scale - 1 );
low       = low + (unsigned short int)
          (( range * s->low_count ) / s->scale );
```

จากนั้นก็เป็นการตรวจสอบบิตลำดับสูงสุดของทั้งค่าต่ำและค่าสูง โดยถ้าหากมีค่าเท่ากับ 1 ให้ทำการเลื่อนบิตนั้นทิ้งไป แต่ถ้าหากบิตดังกล่าวอยู่ในกรณีของการเกิด underflow ให้เลื่อนทิ้งบิตลำดับที่ 2 แทนดังนี้

```
for ( ; ; ) {
   if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
   }
   else if ((low & 0x4000) == 0x4000 && (high & 0x4000) == 0){
      code      ^= 0x4000;
      low       &= 0x3fff;
      high      |= 0x4000;
   } else
      return;
   low    <<= 1;
   high   <<= 1;
   high   |= 1;
   code   <<= 1;
   code   += InputBit( stream );
}
```

ส่วนขั้นตอนสุดท้ายก็เป็นการปรับปรุงโมเดลด้วยรูทีน update_model เช่นเดียวกับการบีบอัดข้อมูล และเมื่ออ่านสายสตริงจากไฟล์ที่ถูกบีบอัดข้อมูลจนหมดก็เป็นอันสิ้นสุดขั้นตอนการขยายขนาดข้อมูล

ในการคอมพายล์หรือสร้างไฟล์ .EXE ผู้อ่านต้องกำหนด memory model ให้เป็นแบบ Compact หรือ Large ก็ได้ เพราะว่าโปรแกรมมีการอ้างหน่วยความจำสำหรับการสร้างตารางค่าความถี่ของ 256 ตาราง เป็นจำนวนมาก ส่วนในตอนต่อไปก็จะมาว่ากันถึงเทคนิคการบีบอัดข้อมูล Dictionary-Based กันต่อ พบกันใหม่ตอนหน้าสวัสดีครับ

ลิสติง ARI1MAIN.H

```
/*
 * ARI1MAIN.H
 */

#ifndef _ARITH1_H
#define _ARITH1_H

void  CompressFile();
void  ExpandFile();

extern char *Usage;
extern char *CompressionName;

#endif  /* _ARI1MAIN_H */
```

ลิสติง ARI1MAIN.C

```
/*
 * ARI1MAIN.C After this is built into a compression
   program of any sort,
 *         the program can be called like this:
 *
 *      arith1 infile outfile option
 *      infile        Input filename.
 *      outfile       Output filename.
 *      option -c     Compress data file.
 *             -e     Expand data file.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bitio.h"
#include "ari1main.h"

void usage_exit();
void print_ratios();
long file_size();

int main( argc, argv )
int argc;
char *argv[];
{
        setbuf( stdout, NULL );
        if ( argc < 3 )
                usage_exit( argv[ 0 ] );

        if ( strcmp( argv[ 3 ], "-c" ) == 0 ) {
        BIT_FILE *output;
        FILE *input;

        input = fopen( argv[ 1 ], "rb" );
        if ( input == NULL )
          fatal_error( "Error opening %s for input\n",
          argv[ 1 ] );
        output = OpenOutputBitFile( argv[ 2 ] );
        if ( output == NULL )
          fatal_error( "Error opening %s for output\n",
          argv[ 2 ] );
        printf( "\nCompressing %s to %s\n", argv[ 1 ],
        argv[ 2 ] );
        printf( "Using %s\n", CompressionName );
        CompressFile( input, output, argc, argv );
        CloseOutputBitFile( output );
        fclose( input );
        print_ratios( argv[ 1 ], argv[ 2 ] );
        return( 0 );
        } else if ( strcmp( argv[ 3 ], "-e" ) == 0 ) {
        FILE *output;
        BIT_FILE *input;

        input = OpenInputBitFile( argv[ 1 ] );
        if ( input == NULL )
          fatal_error( "Error opening %s for input\n",
          argv[ 1 ] );
        output = fopen( argv[ 2 ], "wb" );
        if ( output == NULL )
          fatal_error( "Error opening %s for output\n",
          argv[ 2 ] );
        printf( "\nExpanding %s to %s\n", argv[ 1 ],
        argv[ 2 ] );
        printf( "Using %s\n", CompressionName );
        ExpandFile( input, output, argc, argv );
        CloseInputBitFile( input );
      fclose( output );
      putc( '\n', stdout );
      return( 0 );
    } else
      usage_exit( argv[ 0 ] );
}
```

→

ลิสติง ARI1MAIN.C (ต่อ)

```
void usage_exit( prog_name )
char *prog_name;
{
   char *short_name;
   char *extension;

   short_name = strrchr( prog_name, '\\' );
   if ( short_name == NULL )
      short_name = strrchr( prog_name, '/' );
   if ( short_name == NULL )
      short_name = strrchr( prog_name, ':' );
   if ( short_name != NULL )
      short_name++;
   else
      short_name = prog_name;
   extension = strrchr( short_name, '.' );
   if ( extension != NULL )
      *extension = '\0';
   printf( "\nUsage:  %s %s\n", short_name, Usage );
   exit( 0 );
}

#ifndef SEEK_END
#define SEEK_END 2
#endif

long file_size( name )
char *name;
{
   long eof_ftell;
   FILE *file;

   file = fopen( name, "r" );
   if ( file == NULL )
      return( 0L );
   fseek( file, 0L, SEEK_END );
   eof_ftell = ftell( file );
   fclose( file );
   return( eof_ftell );
}

void print_ratios( input, output )
char *input;
char *output;
{
   long input_size;
   long output_size;
   int ratio;

   input_size = file_size( input );
   if ( input_size == 0 )
      input_size = 1;
   output_size = file_size( output );
   ratio = 100 - (int) ( output_size * 100L / input_size );
   printf( "\nInput bytes:      %ld\n", input_size );
   printf( "Output bytes:     %ld\n", output_size );
   if ( output_size == 0 )
      output_size = 1;
```

ลิสติง ARITH1.C

```
/*
 * The order 1 arithmetic coding module. Compile with ARI1MAIN.C
 * and BITIO.C. With compact or large memory models.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bitio.h"

/* Used to define a symbol in arithmetic coding terms. */
typedef struct {
       unsigned short int low_count;
       unsigned short int high_count;
       unsigned short int scale;
} SYMBOL;

#define MAXIMUM_SCALE   16383  /* Maximum allowed frequency count */
#define END_OF_STREAM   256    /* The EOF symbol */

extern long underflow_bits;    /* To present underflow count in arithmetic coder. */
int *totals[ 257 ];           /* Pointers to the 257 context tables */

/*
 * Local Function prototypes.
 */
void initialize_arithmetic_decoder();
void remove_symbol_from_stream();
void initialize_arithmetic_encoder();
void encode_symbol();
void flush_arithmetic_encoder();
short int get_current_count();

void initialize_model();
void update_model();
void convert_int_to_symbol();
void get_symbol_scale();
int convert_symbol_to_int();

char *CompressionName = "Adaptive order 1 model with arithmetic coding";
char *Usage           = "infile outfile [-c/-e]";
```

ลิสติง
ARITH1.C (ต่อ)

```
/* Compression routine. */
void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int context;

    context = 0;
    initialize_model();
    initialize_arithmetic_encoder();
    for ( ; ; ) {
        c = getc( input );
        if ( c == EOF )
            c = END_OF_STREAM;
        convert_int_to_symbol( c, context, &s );
        encode_symbol( output, &s );
        if ( c == END_OF_STREAM )
            break;
        update_model( c, context );
        context = c;
    }
    flush_arithmetic_encoder( output );
    putchar( '\n' );
}

/* Expansion routine. */
void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int count;
    int c;
    int context;

    context = 0;
    initialize_model();
    initialize_arithmetic_decoder( input );
    for ( ; ; ) {
        get_symbol_scale( context, &s );
        count = get_current_count( &s );
        c = convert_symbol_to_int( count, context, &s );
        remove_symbol_from_stream( input, &s );
        if ( c == END_OF_STREAM )
            break;
        putc( (char) c, output );
        update_model( c, context );
        context = c;
    }
    putchar( '\n' );
}

void initialize_model()
{
    int context;
    int i;

    for ( context = 0 ; context < END_OF_STREAM ; context++ ) {
        totals[ context ] = (int *) calloc( END_OF_STREAM + 2, sizeof(int) );
        if ( totals[ context ] == NULL )
            fatal_error( "Failure allocating context %d", context );
        for ( i = 0 ; i <= ( END_OF_STREAM + 1 ) ; i++ )
            totals[ context ][ i ] = i;
    }
}

void update_model( symbol, context )
int symbol;
int context;
{
    int i;
```

→

ลิสติง
ARITH1.C (ต่อ)

```
    for ( i = symbol + 1 ; i <= ( END_OF_STREAM + 1 ) ; i++ )
        totals[ context ][ i ]++;
    if ( totals[ context ][ END_OF_STREAM + 1 ] < MAXIMUM_SCALE )
        return;
    for ( i = 1 ; i <= ( END_OF_STREAM + 1 ) ; i++ ) {
        totals[ context ][ i ] /= 2;
        if ( totals[ context ][ i ] <= totals[ context ][ i - 1 ] )
            totals[ context ][ i ] = totals[ context ][ i - 1 ] + 1;
    }
}

void convert_int_to_symbol( c, context, s )
int c;
int context;
SYMBOL *s;
{
    s->scale = totals[ context ][ END_OF_STREAM + 1 ];
    s->low_count = totals[ context ][ c ];
    s->high_count = totals[ context ][ c + 1 ];
}

void get_symbol_scale( context, s )
int context;
SYMBOL *s;
{
    s->scale = totals[ context][ END_OF_STREAM + 1 ];
}

int convert_symbol_to_int( count, context, s)
int count;
int context;
SYMBOL *s;
{
    int c;

    for ( c = 0; count >= totals[ context ][ c + 1 ] ; c++ )
        ;
    s->high_count = totals[ context ][ c + 1 ];
    s->low_count = totals[ context ][ c ];
    return( c );
}

/*
 * These four variables define the current state of the arithmetic
 * coder/decoder.
 */
static unsigned short int code;  /* The present input code value       */
static unsigned short int low;   /* Start of the current code range    */
static unsigned short int high;  /* End of the current code range      */
long underflow_bits;             /* Number of underflow bits pending   */

void initialize_arithmetic_encoder()
{
    low = 0;
    high = 0xffff;
    underflow_bits = 0;
}

void flush_arithmetic_encoder( stream )
BIT_FILE *stream;
{
    OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits-- > 0 )
        OutputBit( stream, ~low & 0x4000 );
    OutputBits( stream, 0L, 16 );
}

void encode_symbol( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

    /* These three lines rescale high and low for the new symbol. */
    range = (long) ( high-low ) + 1;
    high = low + (unsigned short int)
```

→

ลิสติง
ARITH1.C (ต่อ)

```
                (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
                (( range * s->low_count ) / s->scale );

    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
            OutputBit( stream, high & 0x8000 );
            while ( underflow_bits > 0 ) {
                OutputBit( stream, ~high & 0x8000 );
                underflow_bits--;
            }
        }
        else if ( ( low & 0x4000 ) && !( high & 0x4000 )) {
            underflow_bits += 1;
            low &= 0x3fff;
            high |= 0x4000;
        } else
            return ;
        low <<= 1;
        high <<= 1;
        high |= 1;
    }
}

short int get_current_count( s )
SYMBOL *s;
{
    long range;
    short int count;

    range = (long) ( high - low ) + 1;
    count = (short int) ((((long) ( code - low ) + 1 ) * s->scale-1 ) / range );
    return( count );
}

void initialize_arithmetic_decoder( stream )
BIT_FILE *stream;
{
    int i;

    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
}

void remove_symbol_from_stream( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

    range = (long)( high - low ) + 1;
    high = low + (unsigned short int)
                (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
                (( range * s->low_count ) / s->scale );

    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
        }
        else if ((low & 0x4000) == 0x4000  && (high & 0x4000) == 0 ) {
            code ^= 0x4000;
            low  &= 0x3fff;
            high  |= 0x4000;
        } else
            return;
        low <<= 1;
        high <<= 1;
        high |= 1;
        code <<= 1;
        code += InputBit( stream );
    }
}
```